НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

до проекту

із дисципліни «Операційні системи»

на тему

Створення microframework WinUI на базі WinAPI

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студент групи КМ-62 | асистент Громова В. В. |
| Антонов М. О. |  |

Київ — 2019

Зміст

[ВСТУП 2](#_Toc4965817)

[1 МЕТА РОБОТИ 3](#_Toc4965818)

[1.1 Мета роботи 3](#_Toc4965819)

[1.2 Поставленні задачі 3](#_Toc4965820)

[2 ОСНОВНА ЧАСТИНА 4](#_Toc4965821)

[2.1 Створення циклу обробки повідомлень 4](#_Toc4965822)

[2.2 Створення класів UI елементів 4](#_Toc4965823)

[3 КЛАС APPLICATION 5](#_Toc4965824)

[3.1 Опис класу Application 5](#_Toc4965825)

[3.2 Public методи класу Application 5](#_Toc4965826)

[3.3 Private поля класу Application 5](#_Toc4965827)

[3 КЛАС WIDGET 6](#_Toc4965828)

[3.1 Опис класу Widget 6](#_Toc4965829)

[3.2 Public методи класу Widget 6](#_Toc4965830)

# ВСТУП

Звіт створений для ознайомлення з процесом створення microframework-у на основі WinAPI. Проект включає в себе створення бібліотеки для побудови графічного інтерфейсу та реалізації функціоналу Windows WinAPI. Також створені програмні засоби на основі створеної бібліотеки WinUI.

У цьому звіті також описаний функціонал саме бібліотеки WinUI. Усі створені класи бібліотеки та їх зовнішні інтерфейси. Окрім цього описана внутрішня реалізація класів бібліотеки, а саме методи та поля класів.

# 1 МЕТА РОБОТИ

### 1.1 Мета роботи

- Створити Framework для зручного використання функціоналу WinAPI.

### 1.2 Поставленні задачі

- Створити ієрархію класів для реалізації об’єктної моделі представлення.

- Створити зручні інтерфейси класів для використання їх у побудові програм.

- Створити з класів статичну бібліотеку на мові С++ для можливості використання бібліотеки.

# 2 ОСНОВНА ЧАСТИНА

### 2.1 Створення циклу обробки повідомлень

Для створення програми використовуючи WinAPI, в першу чергу треба створити цикл обробки повідомлень, які посилає операційна система із черги. Для вирішення цієї задачі було створено клас *Application*, який запускає цикл обробки повідомлень та отримує *HINSTANCE* програми.

### 2.2 Створення класів UI елементів

# 3 КЛАС APPLICATION

### 3.1 Опис класу Application

Клас регулює тривалість життя програми. За допомогою інтерфейсу класу можна розпочинати роботу програми та завершувати.

### 3.2 Public методи класу Application

*Application()* – конструктор класу проводить ініціалізацію поля класу *m\_haccel*, яка призначена для збереження ацелелятору. Ініціалізація проводиться використовуючи функцію *LoadAccelerators*.

*void exec()* – функція створює цикл *while*, який обробляє повідомлення з операційної системи. У циклі зчитується повідомлення типу *MSG,* використовуючи функцію *GetMessage*. Після зчитування повідомлення, воно обробляється та прибирається з черги повідомлень функціями *TranslateMessage* та *DispatchMessage*.

*void quit()* – надає можливість вийти з програми на будь-якому етапі виконання, за допомогою функції *PostQuitMessage*. Функція є статичною тому може бути викладена у будь-якому місці програми.

*static HINSTANCE getInstance()* – статична функція, повертає HINTANCE програми.

### 3.3 Private поля класу Application

*static HINSTANCE instance* – статичне поле класу, яке зберігає *HINSTANCE* програми. Поле класу є статичним, тому до нього може бути наданий доступ з будь якого місця програми, за допомогою функції *getInstance*.

*HACCEL m\_haccel* – Поле типу *HACCEL*. Ініціалізація поля проводиться у конструкторі та зберігає воно дані про ацелелятор. Доступу до цього поля немає.

*MSG m\_msg* – використовується для зберігання повідомлення під час використання функції *GetMessage* у циклі обробки повідомлень.

# 3 КЛАС WIDGET

### 3.1 Опис класу Widget

Абстрактний клас Widget створений для реалізація поліморфізму. Усі графічні елементи бібліотеки є «дітьми» класу Widget, який реалізує спільний для усіх елементів інтерфейс.

### 3.2 Public методи класу Widget

*Widget(WidgetStyle style, Widget\* parent)* – конструктор класу приймає структуру *WidgetStyle*, яка зберігає інформацію про стиль класу вікна та інформацію для створення вікна. Другий параметр є вказівником на батьківський *Widget*.

*void setGeometry(int x, int y, int width, int height)* – метод класу встановлює позицію та розмів елемента відносно родича, якщо він існує. Для реалізації цього процесу використовується функція *WinAPI SetWindowPos(m\_hwnd, NULL, x, y, width, height, NULL)*.

*void setWidth(int width)* – встановлює ширину вікна. Використовується та сама реалізація, що і у *setGeometry,* з різницею у відомих усіх параметрів окрім нової ширини вікна.

*void setHeight(int height)* – встановлює ширину вікна.

*void setMinimumSize(int min\_width, int min\_height)* – встановлює мінімальний розмір елементу. Метод присвоює змінним класу, які відповідають за мінімальний розмір вікна отримані параметри. Реалізації класу можуть використовувати цю інформацію при зміни розміру.

*void setMaximumSize(int wax\_width, int max\_height)* – встановлює максимальний розмір елементу. Реалізація аналогічна до методу *setMinimumSize.*

*void setFixedSize(int width, int height)* – встановлює фіксований розмір елементу. Реалізовано через встановлення поля максимального і мінімального розміру елемента відповідними параметрами. Виходячи с цього елемент не може бути менше або більше вказаних параметрів.

*int x() const* – повертає *x* позицію елемента відносно родича.

*int y() const* – повертає *y* позицію елемента відносно родича.

*int width() const* – повертає ширину елемента.

*int height() const* – повертає висоту елемента.

*Rect size() const* – повертає структуру *Rect*, яка зберігає всі параметри вікна, а саме розмір та положення.

*bool isVisible() const* – метод повертає статус видимості елементу. Якщо елемент видимий повертає *true*, а якщо ні, то *false*.

*HWND getHWND() const* – повертає *HWND(handler)* елементу.

*void addChild(HWND hwnd, Widget\* widget)* – додає «дитину» до елементу. Елемент додається до мапи, у який зберігається *handler* та вказівник до *Widget*.

*void setVisible(bool visible)* – встановлює видимість елементу. Видимість встановлюється функцією *ShowWindow(HWND, int)*, яка належить *WinAPI*. Перший параметр функції *ShowWindow* – handler елементу, а другий визначає статус вікна. Якщо другий параметр *SW\_SHOW*, то вікно стає видимим, а якщо *SW\_HIDE*, то вікно скривається.